

体系结构描述文档

灯具照明行业进销存管理系统



2014-11-3

小百合工作室

# 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 陈云龙 | 2014-11-03 | 初步文档框架 | V0.0 |
| 陈云龙 | 2014-11-04 | 添加数据层分解 | V0.1 |
| 曾婧 | 2014-11-04 | 添加逻辑层初步分解 | V0.2 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[更新历史 1](#_Toc402987533)

[1. 引言 1](#_Toc402987534)

[1.1. 编制目的 1](#_Toc402987535)

[2. 产品概述 1](#_Toc402987536)

[3. 体系结构设计概述 1](#_Toc402987537)

[4. 结构视角 1](#_Toc402987538)

[4.1. 业务逻辑层的分解 1](#_Toc402987539)

[4.1.1. Commoditybl模块 1](#_Toc402987540)

[4.1.2. CommoditySortbl模块 8](#_Toc402987541)

[4.1.3. Inventorybl模块 12](#_Toc402987542)

[4.1.4. Clientbl模块 14](#_Toc402987543)

[4.1.5. Purchasebl模块 16](#_Toc402987544)

[4.1.6. Salebl模块 18](#_Toc402987545)

[4.1.7. Recordbl模块 20](#_Toc402987546)

[4.1.8. Accountbl模块 22](#_Toc402987547)

[4.1.9. AccountBillbl模块 24](#_Toc402987548)

[4.1.10. CashBillbl模块 26](#_Toc402987549)

[4.1.11. Accountainitbl模块 28](#_Toc402987550)

[4.1.12. Approvalbl模块 30](#_Toc402987551)

[4.1.13. Promotionbl模块 32](#_Toc402987552)

[4.1.14. Userbl模块 34](#_Toc402987553)

[4.3. 数据层的分解 37](#_Toc402987554)

[4.3.1. DataFactoryService 37](#_Toc402987555)

[4.3.2. InventoryDataService 40](#_Toc402987556)

[4.3.3. PurchaseDataService 41](#_Toc402987557)

[4.3.4. RecordDataService 42](#_Toc402987558)

[4.3.5. AccountBillDataService 43](#_Toc402987559)

[4.3.6. ApprovalDataService 44](#_Toc402987560)

[4.3.7. UserDataService 45](#_Toc402987561)

[4.3.8. CommodityDataService 46](#_Toc402987562)

[4.3.9. CommoditySortDataService 47](#_Toc402987563)

[4.3.10. SaleDataService 48](#_Toc402987564)

[4.3.11. AccountDataService 49](#_Toc402987565)

[4.3.12. ClientDataService 50](#_Toc402987566)

[4.3.13. AccountaInitDataService 51](#_Toc402987567)

[4.3.14. PromotionDataService 51](#_Toc402987568)

[5. 依赖视角 52](#_Toc402987569)

# 引言

## 编制目的

本报告详细完成灯具照明行业进销存系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

# 产品概述

参考灯具照明行业进销存系统用例文档和灯具照明行业进销存系统软件规格说明中对产品的概括描述。

# 体系结构设计概述

请参考进销存系统需求文档对体系结构设计的概述

# 结构视角

## 业务逻辑层的分解

业务逻辑层的开发包图详见体系结构文档图3。

### Commoditybl模块

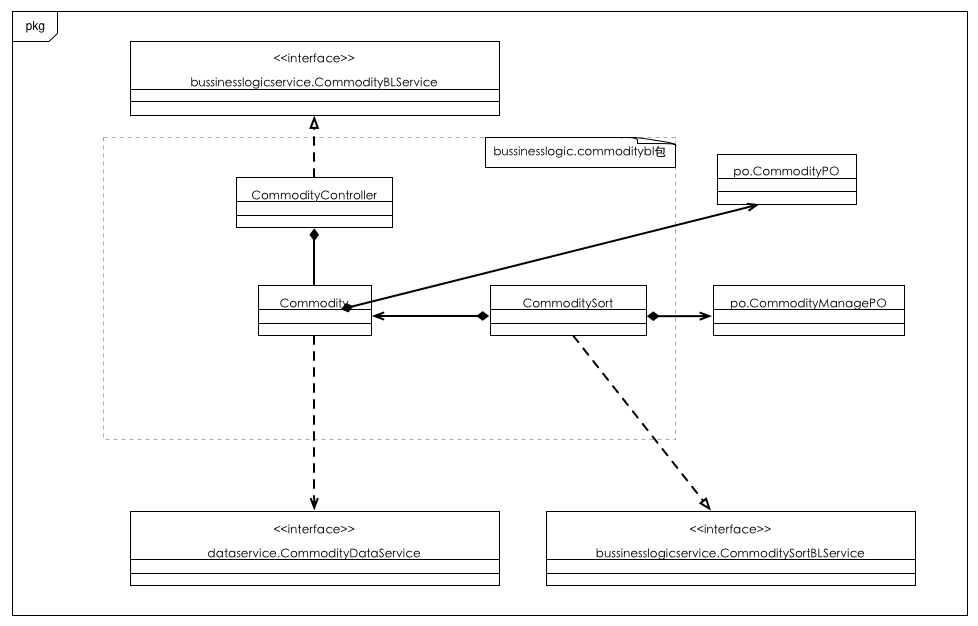
1. 模块概述

commoditybl模块负责实现商品分类和商品的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.CommodityBLService接口。业务逻辑层和数据层之间添加dataservice.CommodityDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommodityController，这样CommodityController会将对商品的逻辑业务处理委托给Commodity对象。CommodityPO是作为商品的持久化对象被添加到设计模型中去的。

Commoditybl模块的设计如图



commoditybl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面所需要的服务 |
| CommodityController | 负责实现商品管理界面所需要的服务 |
| Commodity | 商品的领域模型对象，拥有商品的信息，可以帮助完成商品管理界面所需要的服务 |

1. 模块内部类的接口规范

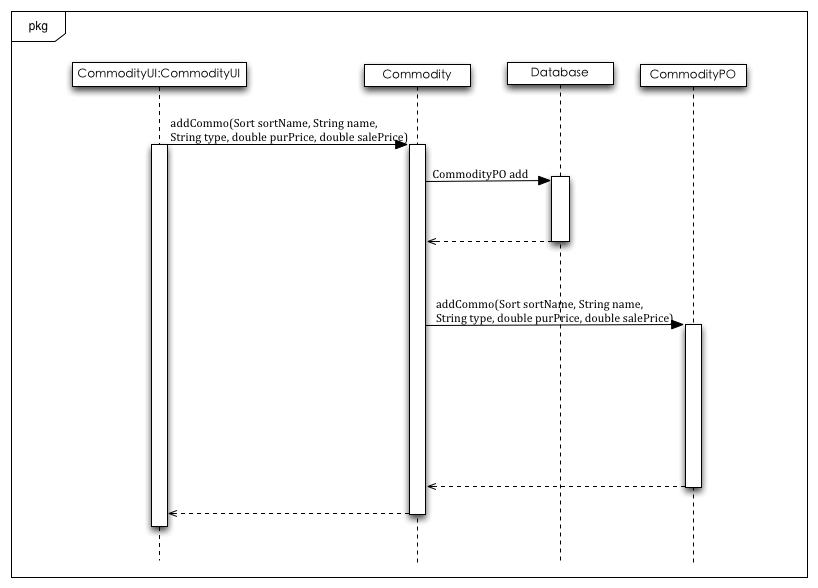
**CommodityController的接口规范**

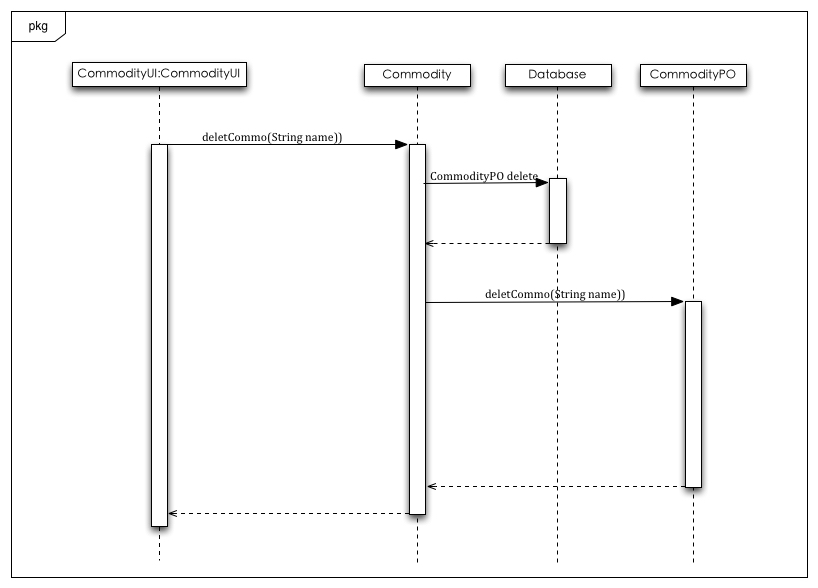
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommodityController.  addCommo | 语法 | public ResultMessage addCommo(Sort sortName, String name, String type, double purPrice, double salePrice) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 将商品添加到商品分类中，持久化更新涉及的对象的数据 |
| CommodityController.  deletCommo | 语法 | public ResultMessage deletCommo  (String name) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 删除该商品，持久化更新涉及的对象的数据 |
| CommodityController.  updCommo | 语法 | public ResultMessage updCommo  (String name, CommoditySort sortName, String type, double purPrice, double salePrice) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 商品更新，持久化更新涉及的对象的数据 |
| Commodity.findCommo | 语法 | public ResultMessage findCommo  (String name, String id, String type) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 查找到该商品 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Commodity.addCommo(Sort sortName, String name, String type, double purPrice, double salePrice) | 加入一个商品 | |
| Commodity.deletCommo  (String name) | 删除一个商品 | |
| Commodity.updCommo  (String name, CommoditySort sortName, String type, double purPrice, double salePrice) | 更改一个商品的信息 | |
| Commodity.findCommo  (String name, String id, String type) | 查找一个商品 | |

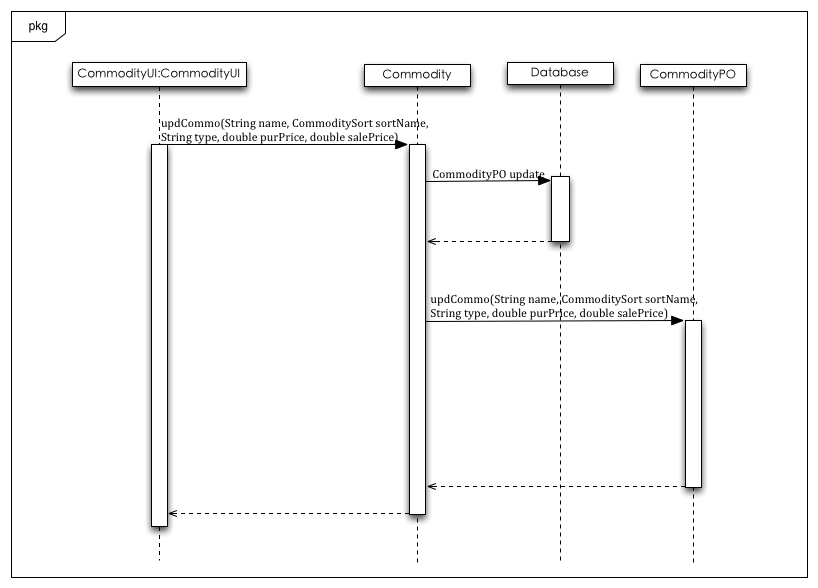
**Commodity的接口规范**

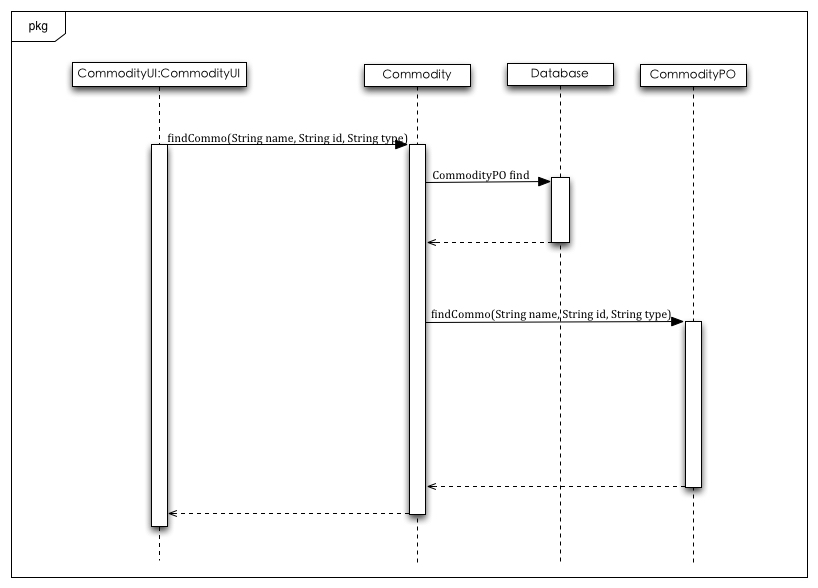
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Commodity.  addCommo | 语法 | public ResultMessage addCommo(Sort sortName, String name, String type, double purPrice, double salePrice) |
| 前置条件 | 分类、商品名称、型号、进价、零售价输入合法 |
| 后置条件 | 将商品添加到商品分类中，持久化更新涉及的对象的数据 |
| Commodity.  deletCommo | 语法 | public ResultMessage deletCommo  (String name) |
| 前置条件 | 商品名称输入合法，商品没有被操作过 |
| 后置条件 | 删除该商品，持久化更新涉及的对象的数据 |
| Commodity.  updCommo | 语法 | public ResultMessage updCommo  (String name, CommoditySort sortName, String type, double purPrice, double salePrice) |
| 前置条件 | 商品名称输入合法，更改信息合法 |
| 后置条件 | 商品更新，持久化更新涉及的对象的数据 |
| Commodity.  findCommo | 语法 | public ResultMessage findCommo  (String name, String id, String type) |
| 前置条件 | 商品名称输入合法，id输入合法 |
| 后置条件 | 查找到该商品 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommodityDataService.  findCommoSort(int id) | 根据ID进行查找单一持久化对象 | |
| CommodityDataService.  findCommo (int id) | 根据ID进行查找单一持久化对象 | |
| CommodityDataService.  findCommos  (Sting field, int id) | 根据字段名和值进行查找多个持久化对象 | |
| CommodityDataService.  add(CommodityPO po) | 添加单一化持久对象 | |
| CommodityDataService.  delete(CommodityPO po) | 删除单一化持久对象 | |
| CommodityDataService.  update(CommodityPO po) | 更改单一化持久对象 | |
| CommodityDataService.  getCommodityData | 得到Commodity数据的服务引用 | |

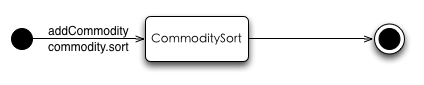
1. 业务逻辑层的动态模型











1. 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### CommoditySortbl模块

1. 模块概述

CommoditySortbl模块负责实现商品分类和商品的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

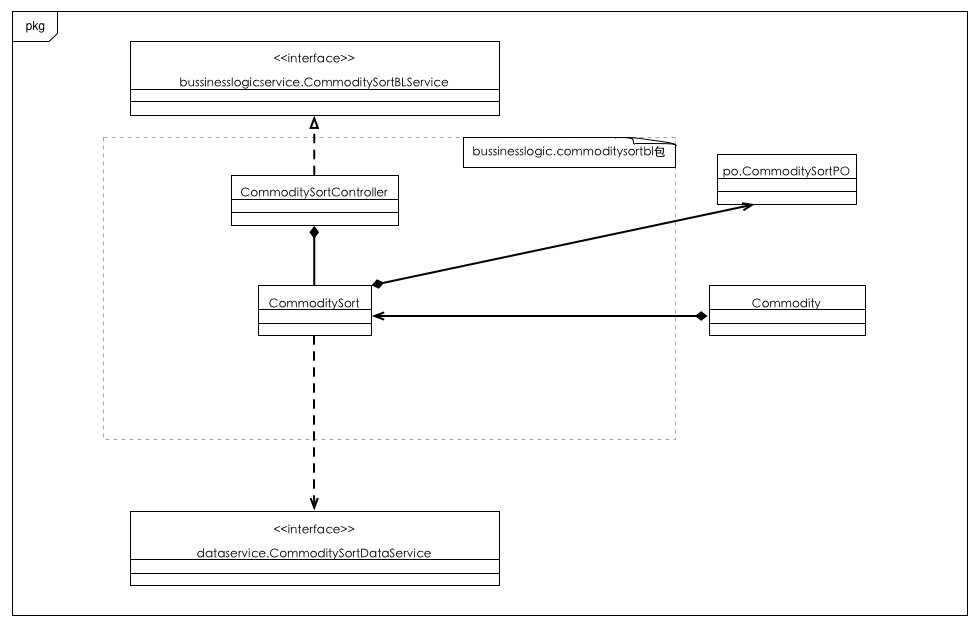
1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.CommoditySortBLService接口。业务逻辑层和数据层之间添加dataservice.CommoditySortDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommoditySortController，这样CommoditySortController会将对商品分类的逻辑业务处理委托给CommoditySort对象。CommoditySortPO是作为商品类别的持久化对象被添加到设计模型中去的。

commoditybl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面所需要的服务 |
| CommoditySortController | 负责实现商品管理界面所需要的服务 |
| CommodityManage | 商品分类的领域模型对象，拥有商品分类的信息，可以帮助完成商品分类界面所需要的服务 |

commoditysortbl模块的设计如图



1. 模块内部类的接口规范

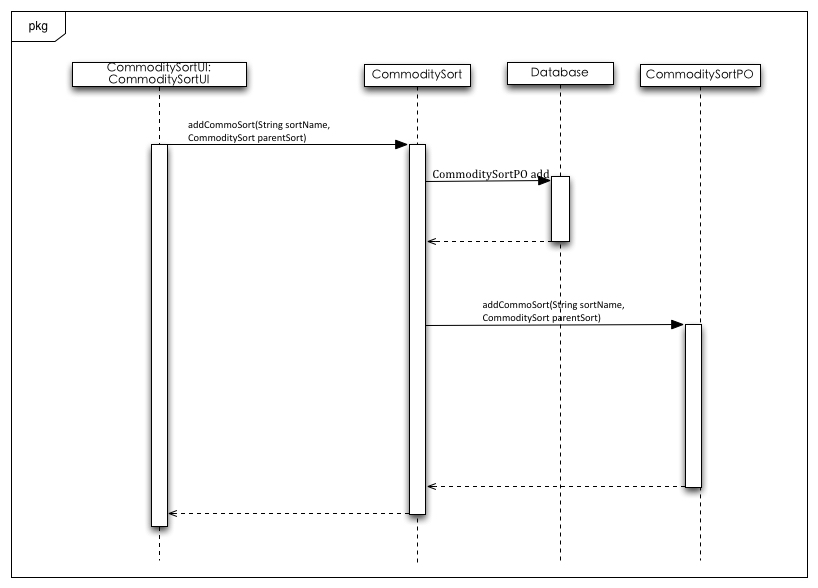
**CommoditySortController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySortController.  addCommoSort | 语法 | public ResultMessage addCommoSort  (String sortName,  CommoditySort parentSort) |
| 前置条件 | 已经创建一个CommodityManager领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity |
| CommoditySortController.  deletCommoSort | 语法 | public ResultMessage deleteCommoSort(  String sortName) |
| 前置条件 | 已经创建一个CommodityManager领域对象，并且输入符合输入规则 |
| 后置条件 | 删除商品分类，持久化更新涉及的对象的数据 |
| CommoditySortController.  updCommoSort | 语法 | public ResultMessage updCommoSort  (Sring sortName) |
| 前置条件 | 已经创建一个CommodityManager领域对象，并且输入符合输入规则 |
| 后置条件 | 更改商品分类，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommoditySort.  addCommoSort  (String sortName,  CommoditySort parentSort) | 加入一个商品分类 | |
| CommoditySort.  deleteCommoSort  (String sortName) | 删除商品分类 | |
| CommoditySort.  updCommoSort  (Sring sortName) | 更改商品分类名称 | |

**Commoditysort的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySort.  addCommoSort | 语法 | public ResultMessage addCommoSort  (String sortName, CommoditySort parentSort) |
| 前置条件 | 分类名称输入合法，并且父类中没有添加商品 |
| 后置条件 | 添加新的商品分类，持久化更新涉及的对象的数据 |
| CommoditySort.  deletCommoSort | 语法 | public ResultMessage deleteCommoSort(  String sortName) |
| 前置条件 | 分类名称输入合法 |
| 后置条件 | 删除商品分类，持久化更新涉及的对象的数据 |
| CommoditySort.  updCommoSort | 语法 | public ResultMessage updCommoSort  (Sring sortName) |
| 前置条件 | 分类名称输入合法 |
| 后置条件 | 更改商品分类，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommoditySortDataService.  findCommoSort(int id) | 根据ID进行查找单一持久化对象 | |
| CommoditySortDataService.  findCommo (int id) | 根据ID进行查找单一持久化对象 | |
| CommoditySortDataService.  findCommos  (Sting field, int id) | 根据字段名和值进行查找多个持久化对象 | |
| CommoditySortDataService.  add(CommodityPO po) | 添加单一化持久对象 | |
| CommoditySortDataService.  delete(CommodityPO po) | 删除单一化持久对象 | |
| CommoditySortDataService.  update(CommodityPO po) | 更改单一化持久对象 | |
| CommoditySortDataService.  getCommodityData | 得到CommoditySort数据的服务引用 | |

1. 业务逻辑层的动态模型





1. 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象

### Inventorybl模块

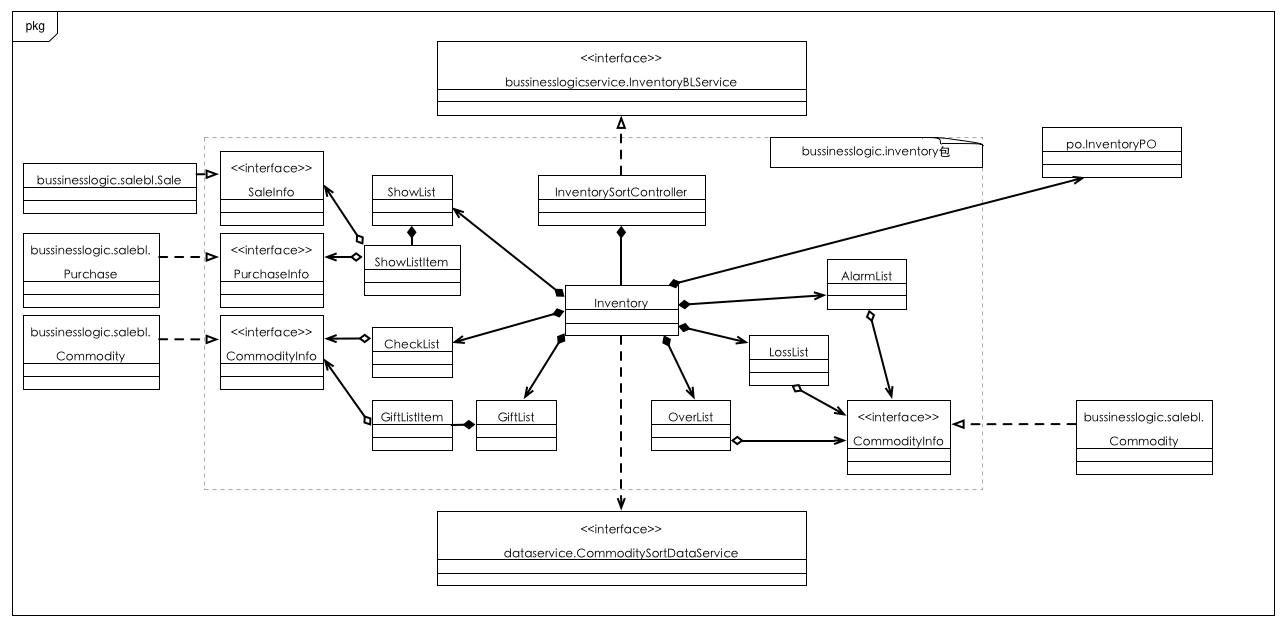
1. 模块概述

Inventorybl模块负责实现仓库的管理，对库存进行报损、报溢，建立赠送单，对库存进行盘点、查看。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. InventoryBLService接口。业务逻辑层和数据层之间添加dataservice. InventoryDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InventoryController，这样InventoryController会将对库存的逻辑业务处理委托给Inventory对象。

Inventorybl模块的设计如图所示



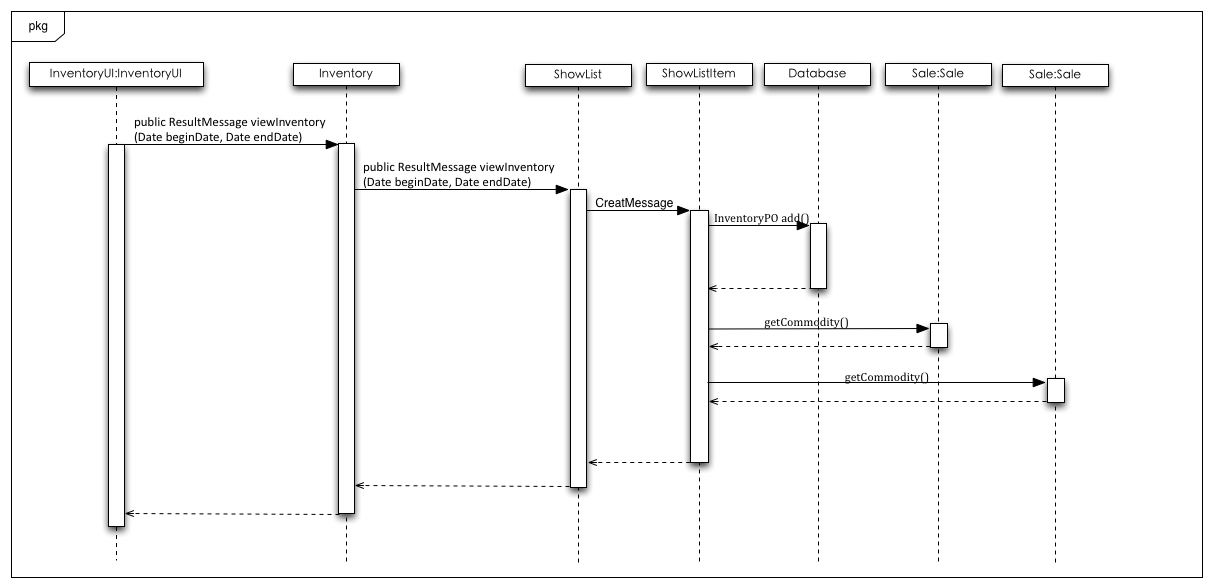
1. 模块内部类的接口规范

**Inventory接口规范**

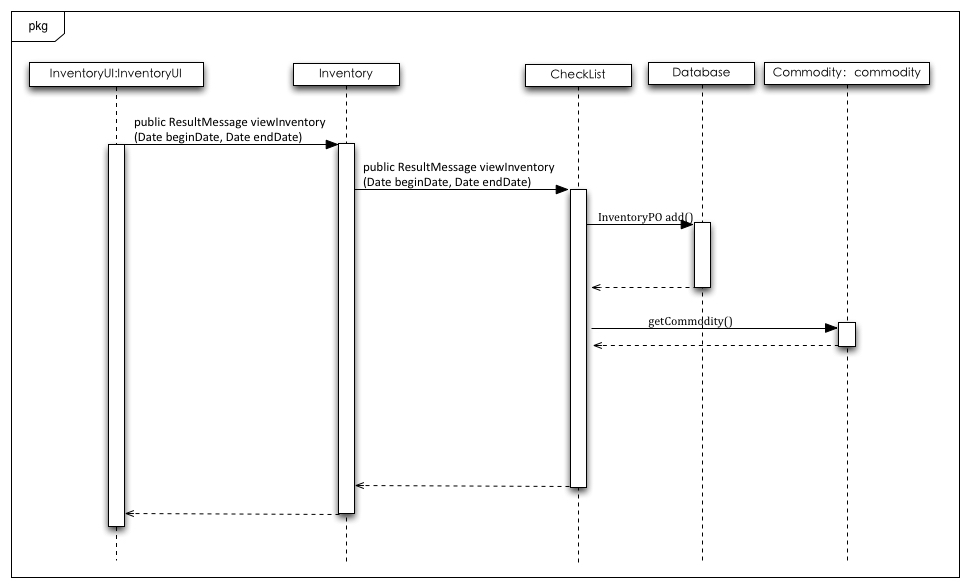
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Inventory.viewInventory | 语法 | public ResultMessage viewInventory  (Date beginDate, Date endDate) |
| 前置条件 | 日期的输入合法 |
| 后置条件 | 显示在此时间段内的出／入库数量／金额，销售／进货数量／金额，以及库存合计 |
| Inventory.checkRecord | 语法 | public ResultMessage checkRecord( ) |
| 前置条件 | 无 |
| 后置条件 | 显示截至盘点时间的库存快照 |
| Inventory.giftRecord | 语法 | public ResultMessage giftRecord  (CommodityVO commodity,  int commoNum, Client client) |
| 前置条件 | 商品、商品数量、客户的输入合法 |
| 后置条件 | 生成相应的库存赠送单 |
| Inventory.overflowRecord | 语法 | public ResultMessage overflowRecord  (CommodityVO commodity, int num) |
| 前置条件 | 完成库存的盘点，输入数据 |
| 后置条件 | 生成相应的库存报溢单 |
| Inventory.lossRecord | 语法 | public ResultMessage lossRecord  (CommodityVO commodity, int num) |
| 前置条件 | 完成库存的盘点，输入数据 |
| 后置条件 | 生成相应的库存报损单 |
| Inventory.alarmRecord | 语法 | public ResultMessage alarmRecord  (CommodityVO commodity, int num) |
| 前置条件 | 进行商品数量上的操作 |
| 后置条件 | 生成相应的库存报警单 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| InventoryDataService.  insert(InventoryBillPO po) | 生成单据，告知是否成功 | |

1. 业务逻辑层的动态模型

如图表明了当库存管理人员输入需要查看的时间段之后，库存业务逻辑处理的相关对象之间的协作



如图表明了当库存管理人员想要添加赠送单的时候的顺序图



如图所示的状态图描述了Inventory对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。

### Clientbl模块

1. 模块概述

Clientbl模块负责客户的管理，创建新的客户，删除客户，修改客户，查找客户。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. ClientBLService接口。业务逻辑层和数据层之间添加dataservice. ClientDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ClientController，这样ClientController会将对客户的逻辑业务处理委托给Client对象。ClientPO是作为客户的持久化对象被添加到设计模型中去的。

1. 模块内部类的接口规范

**Client的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Client.findClient | 语法 | public ResultMessage findClient(String keywords, long id) |
| 前置条件 | 关键字符合输入条件，id符合输入条件 |
| 后置条件 | 查找到目标客户 |
| Client.addClient | 语法 | public ResultMessage addClient(String kind, String name, String level, String phoneNum, String address, String post, String email, UserVO salesman) |
| 前置条件 | 类型、名字、级别、电话、地址、邮编符合输入条件 |
| 后置条件 | 添加客户，持久化更新涉及的对象的数据 |
| Client.updClient | 语法 | public ResultMessage updClient(String kind, String name, String level, String phoneNum, String address, String email, String post, UserVO salesman) |
| 前置条件 | 修改的属性符合输入条件 |
| 后置条件 | 客户属性被修改，持久化更新涉及的对象的数据 |
| Client.deletClient | 语法 | public ResultMessage deletClient(String name) |
| 前置条件 | 输入的客户名称符合输入条件 |
| 后置条件 | 删除该客户，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| ClientDataService.  findClient(int id) | 根据ID进行查找单一持久化对象 | |
| ClientDataService.  findClients(String fields, int id) | 根据字段名和值进行查找多个持久化对象 | |
| ClientDataService.  add(ClientPO po) | 添加单一化持久对象 | |
| ClientDataService.  delet(ClientPO po) | 删除单一化持久对象 | |
| ClientDataService.  update(ClientPO po) | 更改单一化持久对象 | |
| ClientDataService.  getClientData | 得到Client数据的服务引用 | |

1. 业务逻辑层的动态模型

### Purchasebl模块

1. 模块概述

Purchasebl模块负责进货单、进货退货单的创建。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. PurchaseBLService接口。业务逻辑层和数据层之间添加dataservicePurchaseDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PurchaseController，这样PurchaseController会将对进货的逻辑业务处理委托给Purchase对象。PurchasePO是作为进货单的持久化对象被添加到设计模型中去的。

1. 模块内部类的接口规范

**Purchase的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Purchase.getID | 语法 | public ResultMessage getID () |
| 前置条件 | 启动一个创建进货单回合 |
| 后置条件 | 得到该销售单的编号 |
| Purchase.addCommodities | 语法 | public ResultMessage addCommodities(String name, int num, int price, String remark) |
| 前置条件 | 商品名称、数量、单价、备注符合输入规则 |
| 后置条件 | 添加商品到商品清单 |
| Sale.getBeforePrice | 语法 | public ResultMessage getBeforePrice  (ArrayList<CommodityItemVO> commodities) |
| 前置条件 | 要销售的商品已经添加商品列表 |
| 后置条件 | 得到所有商品的总价 |
| Purchase.addPur | 语法 | public ResultMessage addSale(String ID, ClientVO client, UserVO user, Storage storage, ArrayList<CommodityItemVO> commodities, int sumPrice, String remark) |
| 前置条件 | 商品、数量、单价、备注符合输入规则 |
| 后置条件 | 制定出新的进货单 |
| Purchase.addPurBack | 语法 | public ResultMessage addSaleBack(String ID, ClientVO client, UserVO saleman, UserVO user, Storage storage, ArrayList<CommodityItemVO> commodities, int sumPrice) |
| 前置条件 | 商品、数量、单价、备注符合输入规则 |
| 后置条件 | 制定出新的进货退货单 |
| Purchase.submit | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PurchaseDataService.  find(String id) | 根据ID进行查找单一持久化对象 | |
| PurchaseDataService.  insert(PurchasePO po) | 插入单一持久化对象 | |
| PurchaseDataService.  delete(PurchasePO po) | 删除单一持久化对象 | |
| PurchaseDataService.  update(PurchasePO po) | 更新单一持久化对象 | |
| PurchaseDataService.init | 清空单一持久化对象 | |

1. 业务逻辑层的动态模型

### Salebl模块

1. 模块概述

Salebl模块负责销售单、销售退货单的创建。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. SaleBLService接口。业务逻辑层和数据层之间添加dataservice. SaleDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了SaleController，这样SaleController会将对销售的逻辑业务处理委托给Sale对象。SalePO是作为销售单的持久化对象被添加到设计模型中去的。

1. 模块内部类的接口规范

**Sale接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Sale.getID | 语法 | public ResultMessage getID () |
| 前置条件 | 启动一个创建销售单回合 |
| 后置条件 | 得到该销售单的编号 |
| Sale.addCommodities | 语法 | public ResultMessage addCommodities(String name, int num, int price, String remark) |
| 前置条件 | 商品名称、数量、单价、备注符合输入规则 |
| 后置条件 | 添加商品到商品清单 |
| Sale.getBeforePrice | 语法 | public ResultMessage getBeforePrice  (ArrayList<CommodityItemVO> commodities) |
| 前置条件 | 要销售的商品已经添加商品列表 |
| 后置条件 | 得到所有商品的总价 |
| Sale.getAfterPrice | 语法 | public ResultMessage getAfterPrice  (int beforePrice, int allowance, int voucher) |
| 前置条件 | 折让前总额、折让、代金券金额符合输入规则 |
| 后置条件 | 得到折让后的总额 |
| Sale.addSale | 语法 | public ResultMessage addSale(String ID, ClientVO client, UserVO saleman, UserVO user, Storage storage, ArrayList<CommodityItemVO> commodities, int beforePrice, int allowance, int voucher, String remark , int afterPrice) |
| 前置条件 | 商品、数量、单价、备注符合输入规则 |
| 后置条件 | 制定出新的销售单 |
| Sale.addSaleBack | 语法 | public ResultMessage addSaleBack(String ID, ClientVO client, UserVO saleman, UserVO user, Storage storage, ArrayList<CommodityItemVO> commodities, int beforePrice, int allowance, int voucher, int afterPrice, String remark) |
| 前置条件 | 商品、数量、单价、备注符合输入规则 |
| 后置条件 | 制定出新的销售退货单 |
| Sale.submit | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| SaleDataService.  find(String id) | 根据ID进行查找单一持久化对象 | |
| SaleDataService.  insert(PurchasePO po) | 插入单一持久化对象 | |
| SaleDataService.  delete(PurchasePO po) | 删除单一持久化对象 | |
| SaleDataService.  update(PurchasePO po) | 更新单一持久化对象 | |
| SaleDataService.init | 清空单一持久化对象 | |

1. 业务逻辑层的动态模型

### Recordbl模块

1. 模块概述

Recordbl模块负责进行销售明细表、经营情况表、经营历程表的查看。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. RecordBLService接口。业务逻辑层和数据层之间添加dataservice. RecordDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了RecordController，这样RecordController会将对报表的逻辑业务处理委托给Record对象。

1. 模块内部类的接口规范

**Record的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Record.saleDetail | 语法 | public ResultMessage saleDetail(Date begin, Date end, CommodityVO commodity, ClientVO client, UserVO salesman, Storage Storage); |
| 前置条件 | 时间区间、商品名、客户、业务员、仓库符合输入规则 |
| 后置条件 | 返回符合条件的销售明细表 |
| Record.businessPro | 语法 | public ResultMessage bussinessPro  (Date begin, Date end, BillType billType, ClientVO client, UserVO salesman, Storage storage); |
| 前置条件 | 时间区间、单据类型、商品、业务员、仓库符合输入规则 |
| 后置条件 | 返回符合条件的单据的历程表 |
| Record.red | 语法 | public ResultMessage red  (ValueObject valueRecord) |
| 前置条件 | 有之前的单子做模板 |
| 后置条件 | 添加红冲 |
| Record.copyRed | 语法 | public ResultMessage copyRed  (ValueObject valueRecord) |
| 前置条件 | 有之前的单子做模板 |
| 后置条件 | 修改红冲 |
| Record.bussinessState | 语法 | public ResultMessage businessState  (Date begin, Date end) |
| 前置条件 | 时间区间符合输入规则 |
| 后置条件 | 返回符合条件的经营情况表 |
| Record.earning | 语法 | public ResultMessage earning() |
| 前置条件 | 完成单据的类型选择 |
| 后置条件 | 返回计算得到的收入 |
| Record.expense | 语法 | public ResultMessage expense() |
| 前置条件 | 完成单据的类型选择 |
| 后置条件 | 返回计算得到的支出 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| RecordDataService.  insert(RecordPO po) | 插入单一持久化对象 | |
| RecordDataService.  delete(RecordPO po) | 删除单一持久化对象 | |
| RecordDataService.  update(RecordPO po) | 更新单一持久化对象 | |

1. 业务逻辑层的动态模型

### Accountbl模块

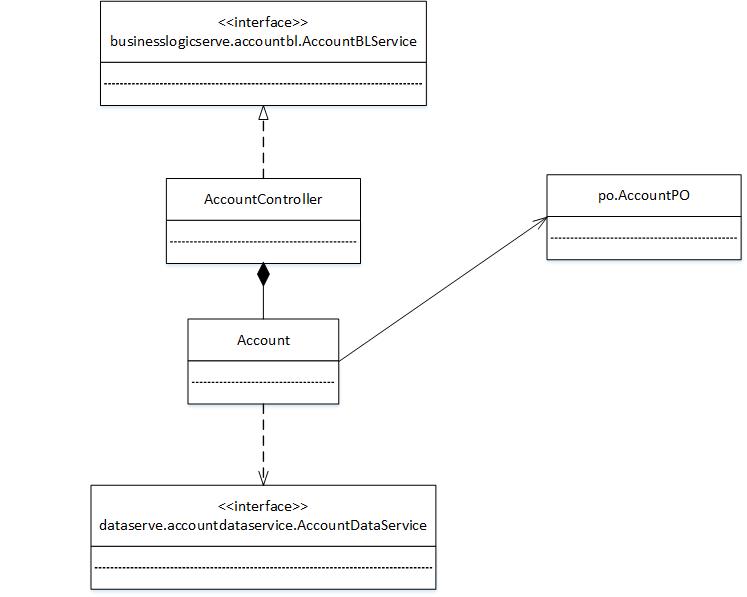
1. 模块概述

Accountbl模块负责进行公司账户的管理，对账户进行创建、更改、删除、查找。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. AccountBLService接口。业务逻辑层和数据层之间添加dataservice. AccountDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountController，这样AccountController会将对账户的逻辑业务处理委托给Account对象。AccountPO是作为账户的持久化对象被添加到设计模型中去的。

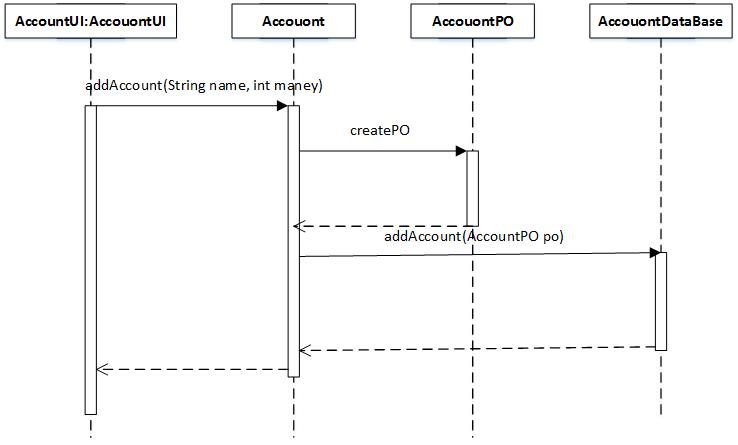
accountbl模块的设计如图



1. 模块内部类的接口规范

**Account的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Account.findAccount | 语法 | public ResultMessage findAccount  (String keyword) |
| 前置条件 | keyword符合输入规则 |
| 后置条件 | 返回查找的账户 |
| Account.addAccount | 语法 | public ResultMessage addAccount(String name, int money) |
| 前置条件 | name和money符合输入规则 |
| 后置条件 | 增加新的账户，持久化更新涉及的对象的数据 |
| Account.deletAccount | 语法 | public ResultMessage deleteAccount  (String name) |
| 前置条件 | name符合输入规则 |
| 后置条件 | 删除账户，持久化更新涉及的对象的数据 |
| Account.updAccount | 语法 | public ResultMessage updAccount(String name) |
| 前置条件 | name符合输入规则 |
| 后置条件 | 更改账户，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountDataService.  find(String id) | 根据ID进行查找单一持久化对象 | |
| AccountDataService.  insert(AccountPO po) | 插入单一持久化对象 | |
| AccountDataService.  delete(AccountPO po) | 删除单一持久化对象 | |
| AccountDataService.  update(AccountPO po) | 更新单一持久化对象 | |
| AccountDataService.init() | 清空单一持久化对象 | |
| AccountDataService.getClientData | 得到Account数据的服务引用 | |

1. 业务逻辑层的动态模型

下图是添加账户的顺序图和状态图



添加账户状态图

### AccountBillbl模块

1. 模块概述

AccountBillbl模块负责建立收款单、付款单。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. AccountBillBLService接口。业务逻辑层和数据层之间添加dataservice. AccountBillDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountBillController，这样AccountBillController会将对收付款的逻辑业务处理委托给AccountBill对象。AccountBillPO是作为收付款单的持久化对象被添加到设计模型中去的。

1. 模块内部类的接口规范

**AccountBill的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountBill.addBillItem | 语法 | public ResultMessage addBillItem(AccountVO account, int money, String remark) |
| 前置条件 | 账户名、转账金额、备注符合输入规则 |
| 后置条件 | 建立转账列表 |
| AccountBill.getSumMoney | 语法 | public ResultMessage getSumMoney  (ArrayList<BillItemVO> accounts); |
| 前置条件 | 已经建立转账列表 |
| 后置条件 | 得到总金额 |
| AccountBill.addExpenses | 语法 | public ResultMessage addExpenses  (String id, ClientVO client, UserVO user, ArrayList<BillItemVO> bills, int sumMoney); |
| 前置条件 | 编号、客户、账户、总金额、备注符合输入规则 |
| 后置条件 | 建立收款单 |
| AccountBill.addPay | 语法 | public ResultMessage addPay(String id, ClientVO client, UserVO user, ArrayList<BillItemVO> bills, int sumMoney) |
| 前置条件 | 编号、客户、操作员、转账列表、金额、备注符合输入规则 |
| 后置条件 | 建立付款单 |
| AccountBill.submit | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountBillDataService.  find(String id) | 根据查找单一持久化对象 | |
| AccountBillDataService.  insert(FuntPO po) | 插入单一持久化对象 | |
| AccountBillDataService.  delete(FuntPO po) | 删除单一持久化对象 | |
| AccountBillDataService.  update(FuntPO po) | 更新单一持久化对象 | |
| AccountBillDataService.init | 初始化单一持久化对象 | |

1. 业务逻辑层的动态模型

### CashBillbl模块

1. 模块概述

CashBillbl模块负责建立现金费用单具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. CashBillBLService接口。业务逻辑层和数据层之间添加dataservice. CashBillDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CashBillController，这样CashBillController会将对现金单的逻辑业务处理委托给CashBill对象。CashBill是作为现金单的持久化对象被添加到设计模型中去的。

1. 模块内部类的接口规范

**CashBill的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| cashBill.addBillItem | 语法 | public ResultMessage addBillItem(String name, int money, String remark) |
| 前置条件 | 账户名、转账金额、备注符合输入规则 |
| 后置条件 | 建立转账列表 |
| cashBill.getSumMoney | 语法 | public ResultMessage getSumMoney  (ArrayList< CashItemVO > bills); |
| 前置条件 | 已经建立转账列表 |
| 后置条件 | 得到总金额 |
| cashBill. addCashBill | 语法 | public ResultMessage addCashBill (String id, UserVO user, AccountVO account, ArrayList<CashItemVO> bills, int sumMoney); |
| 前置条件 | 编号、客户、账户、总金额、备注符合输入规则 |
| 后置条件 | 建立收款单 |
| cashBill.submit | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CashBillDataService.  find(String id) | 根据查找单一持久化对象 | |
| CashBillDataService.  insert(CashBillPO po) | 插入单一持久化对象 | |
| CashBillDataService.  delete(CashBillPO po) | 删除单一持久化对象 | |
| CashBillDataService.  update(CashBillPO po) | 更新单一持久化对象 | |
| CashBillDataService.init | 初始化单一持久化对象 | |

1. 业务逻辑层的动态模型

### Accountainitbl模块

1. 模块概述

Accountainitbl模块负责实现建账界面所需要的服务,具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. AccountainitBLService接口。业务逻辑层和数据层之间添加dataservice. AccountainitDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountainitController，这样AccountainitController会将对期初建账的逻辑业务处理委托给Accountainit对象。AccountainitPO是作为期初建账的持久化对象被添加到设计模型中去的。

Accountainitbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountainitController | 负责实现期初建账界面所需要的服务 |
| Accountainit | 期初建账的领域模型对象，拥有期初建账的信息，可以帮助完成期初建账界面所需要的服务 |

1. 模块内部类的接口规范

**AccountainitController接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountainitController.addInit | 语法 | public ResultMessage addInit  (ArrayList<CommodityVO> commodities,  ArrayList<ClientVO> clients,  ArrayList<AccountVO> accounts) |
| 前置条件 | 已经创建一个Accountainit领域对象,商品、客户、银行账户符合输入规范 |
| 后置条件 | 调用Accountainit的addInit方法 |
| AccountainitController.getEverage | 语法 | public ResultMessage getEverage  (int time, double purPrice, double salePrice) |
| 前置条件 | 已经创建一个Accountainit领域对象,已经添加了商品信息 |
| 后置条件 | 调用Accountainit的getEverage方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Accountainit.addInit (AccountaInitPO po) | 插入单一持久化对象 | |
| Accountainit. getEverage | 返回上年的平均进价和售价 | |

**Accountainit接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountaInit.addInit | 语法 | public ResultMessage addInit  (ArrayList<CommodityVO> commodities,  ArrayList<ClientVO> clients,  ArrayList<AccountVO> accounts) |
| 前置条件 | 商品、客户、银行账户符合输入规范 |
| 后置条件 | 建立期初账目 |
| AccountaInit.getEverage | 语法 | public ResultMessage getEverage  (int time, double purPrice, double salePrice) |
| 前置条件 | 已经添加了商品信息 |
| 后置条件 | 返回上年的平均进价和售价 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountaInitDataService.  insert(AccountaInitPO po) | 插入单一持久化对象 | |
| AccountaInitDataService.  init | 初始化持久数据 | |

### Approvalbl模块

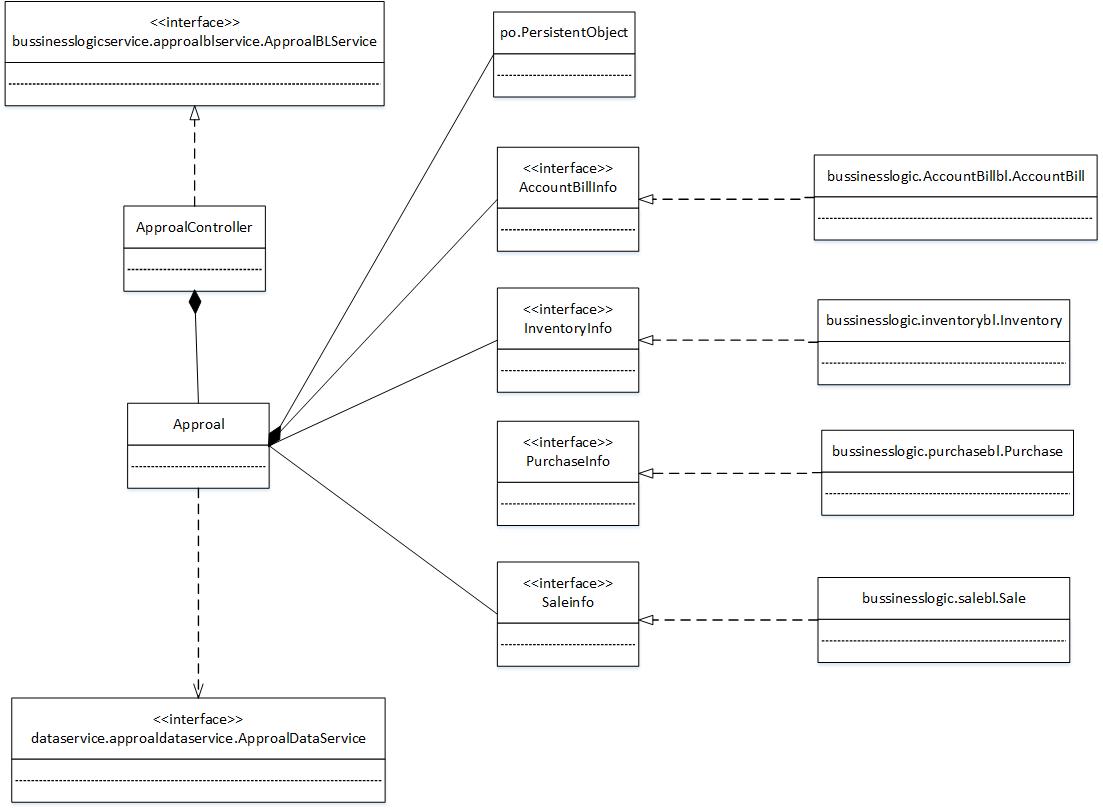
1. 模块概述

Approalbl模块负责实现审批界面所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. ApproalBLService接口。业务逻辑层和数据层之间添加dataservice. ApproalDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ApproalController，这样ApproalController会将对促销的逻辑业务处理委托给Approal对象。

Approalbl模块的设计如图



Approalbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| ApproalController | 负责实现审批管理界面所需要的服务 |
| Approal | 审批的领域模型对象，拥有审批单子的信息，可以帮助完成审批管理界面所需要的服务 |

1. 模块内部类的接口规范

**ApproalController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ApproalController.checkRecord | 语法 | public ResultMessage checkRecord  (PersistentObject record) |
| 前置条件 | 已经创建一个Approal领域对象,有需要审核的单据 |
| 后置条件 | 调用Approal领域对象的checkRecord方法 |
| ApproalController.updRecord | 语法 | public ResultMessage updRecord  (PersistentObject record) |
| 前置条件 | 已经创建一个Approal领域对象,已经查看到单据内容 |
| 后置条件 | 调用Approal领域对象的updRecord方法 |
| ApproalController.submitRecord | 语法 | public ResultMessage submitRecord  (PersistentObject record) |
| 前置条件 | 已经创建一个Approal领域对象,确认单据内容没有错误 |
| 后置条件 | 调用Approal领域对象的submitRecord方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Approal. checkRecord | 查看单据内容 | |
| Approal. updRecord | 修改单据内容 | |
| Approal. submitRecord | 确认单据内容没有错误 | |

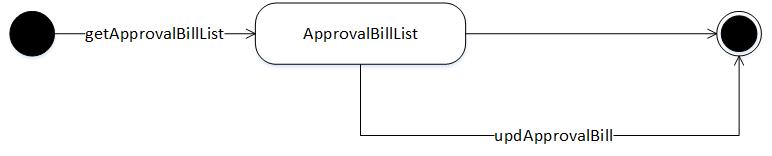
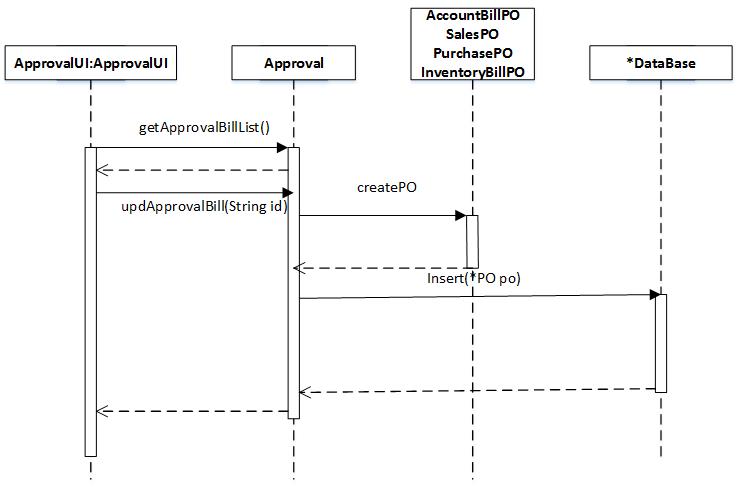
**Approal的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Apporval.checkRecord | 语法 | public ResultMessage checkRecord  (PersistentObject record) |
| 前置条件 | 有需要审核的单据 |
| 后置条件 | 查看单据内容 |
| Apporval.updRecord | 语法 | public ResultMessage updRecord  (PersistentObject record) |
| 前置条件 | 已经查看到单据内容 |
| 后置条件 | 修改单据内容 |
| Apporval.submitRecord | 语法 | public ResultMessage submitRecord  (PersistentObject record) |
| 前置条件 | 确认单据内容没有错误 |
| 后置条件 | 通过单据，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| ApprovalDataService.  Insert(ApprovalPO po) | 插入单一持久化对象 | |

1. 业务逻辑层的动态模型

下图为审批的顺序图和状态图。

getApprovalBill被ui调用，然后进入界面进入显示ApprovalBillList状态，可以审批并更新单据或者直接结束审批。



### Promotionbl模块

1. 模块概述

Promotionbl模块负责实现对促销策略的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. PromotionBLService接口。业务逻辑层和数据层之间添加dataservice. PromotionDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PromotionController，这样PromotionController会将对促销的逻辑业务处理委托给Promotion对象。PromotionPO是作为促销策略的持久化对象被添加到设计模型中去的。

Promotionbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| PromotionController | 负责实现促销策略管理界面所需要的服务 |
| Promotion | 促销策略的领域模型对象，拥有促销策略的信息，可以帮助完成促销策略管理界面所需要的服务 |

1. 模块内部类的接口规范

**PromotionController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionController.promote | 语法 | public ResultMessage promote() |
| 前置条件 | 已经创建一个Promotion领域对象，有一些基本数据，比如商品、销售数据 |
| 后置条件 | 调用Promotion领域对象的promote方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Promotion.find(String condition) | 得到销售策略 | |

**Promotion的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Promotion.promote | 语法 | public ResultMessage promote() |
| 前置条件 | 有一些基本数据，比如商品、销售数据 |
| 后置条件 | 得到销售策略 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionDataService.  find(String condition) | 根据需要制定的策略条件查找可以生成销售策略的持久化对象 | |

### Userbl模块

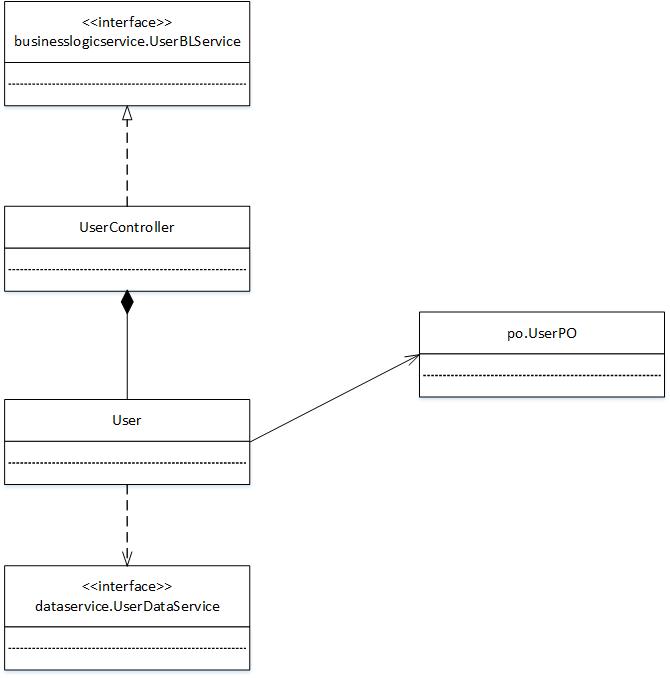
1. 模块概述

Userbl模块负责实现用户的管理具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.UserBLService接口。业务逻辑层和数据层之间添加dataservice.UserDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserController会将对商品的逻辑业务处理委托给User对象。UserPO是作为客户的持久化对象被添加到设计模型中去的。

Userbl模块的设计如图



Userbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责实现用户管理界面所需要的服务 |
| User | 用户的领域模型对象，拥有用户的信息，可以帮助完成用户管理界面所需要的服务 |

1. 模块内类接口规范

**UserController的接口规范**

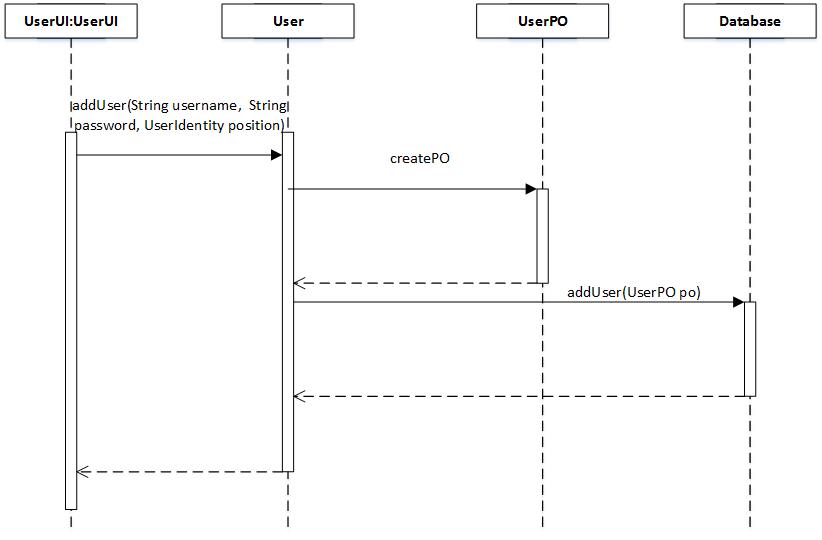
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | 已创建一个User领域对象，password符合输入规则 |
| 后置条件 | 调用User领域的login方法 |
| UserController.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 已创建一个User领域对象，添加的User不重复 |
| 后置条件 | 调用User领域的addUser方法 |
| UserController.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | 已创建一个User领域对象，id符合输入规则 |
| 后置条件 | 调用User领域的deleteUser方法 |
| UserController.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 已创建一个User领域对象，用户名、密码、职位符合输入规则 |
| 后置条件 | 调用User领域的updateUser方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| UserController.login | 用户登录 | |
| UserController.addUser | 添加一个用户 | |
| UserController.deletUser | 删除一个用户 | |
| UserController.updateUser | 更新一个用户 | |

**User接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | password符合输入规则 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回登录验证的结果 |
| User.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 添加的User不重复 |
| 后置条件 | 添加User |
| User.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id删除该用户 |
| User.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 用户名、密码、职位符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id，更改该用户的权限 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getUserDataBase | 得到User的服务的引用 | |
| UserDateService.find  (UserPO po) | 查找单一持久化对象 | |
| UserDataService.insert  (UserPO po) | 插入单一持久化对象 | |
| UserDataService.delete  (UserPO po) | 删除单一持久化对象 | |
| userDataService.update  (UserPO po) | 更新单一持久化对象 | |

1. 业务逻辑层的动态模型

下图表明了灯具管理系统中，添加一个用户时，用户业务逻辑处理的相关对象之间的协作。然后是添加用户的状态图



添加用户顺序图



User状态图

1. 用户逻辑层的设计原理

利用委托式的控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

## 数据层的分解

数据层的开发包图请参见体系结构描述文档图4。

### DataFactoryService

1. 模块概述

DataFactoryService模块是一个工厂类，用来创建其他的DataService实例，并将其统一管理起来。

1. 整体结构

DataFactoryService模块设计如下图所示



1. 模块内部的接口规范

DataFactory内部的的接口规范如下表

**DataFactory的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| DataFactory.getAccountaInitData | 语法 | **public DataService getAccountaInitData() throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**AccountaInitData**对象 |
| DataFactory. getAccountData | 语法 | **public DataService getAccountData() throws RemoteException;** |
| 前置条件 | 无 |
| 后置条件 | 返回**AccountData**对象 |
| DataFactory. getApprovalData | 语法 | **public DataService getApprovalData() throws RemoteException;** |
| 前置条件 | 无 |
| 后置条件 | 返回**ApprovalData**对象 |
| DataFactory. getClientData | 语法 | **public DataService getClientData() throws RemoteException;** |
| 前置条件 | 无 |
| 后置条件 | 返回**ClientData**对象 |
| DataFactory. getCommodityData | 语法 | **public DataService getCommodityData() throws RemoteException;** |
| 前置条件 | 无 |
| 后置条件 | 返回**CommodityData**对象 |
| DataFactory.  getCommoditySortData | 语法 | **public DataService getCommoditySortData () throws RemoteException;** |
| 前置条件 | 无 |
| 后置条件 | 返**回CommoditySortData**对象 |
| DataFactory. getAccountBillData | 语法 | **public DataService getAccountBillData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**AccountBillData**对象 |
| DataFactory. getInventoryData | 语法 | **public DataService getInventoryData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**InventoryData**对象 |
| DataFactory. getPromotionData | 语法 | **public DataService getPromotionData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**PromotionData**对象 |
| DataFactory. getPurchaseData | 语法 | **public DataService getPurchaseData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**PurchaseData**对象 |
| DataFactory. getRecordData | 语法 | **public DataService getRecordData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**RecordData**对象 |
| DataFactory. getSaleData | 语法 | **public DataService getSaleData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**SaleData**对象 |
| DataFactory. getUserData | 语法 | **public DataService getUserData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**UserData**对象 |
| DataFactory. getCashBillData | 语法 | **public DataService getCashBillData () throws RemoteException** |
| 前置条件 | 无 |
| 后置条件 | 返回**CashBillData**对象 |

### InventoryDataService

1. 模块概述

InventoryDataService模块提供库存单据的提交服务（赠送单、报溢单、报损单、报警单）

1. 模块内类接口规范

InventoryDataService接口规范如下表

**InventoryDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| InventoryDataService.insert | 语法 | public ResultMessage insert(InventoryBillPO po) |
| 前置条件 | 无 |
| 后置条件 | 返回提交是否成功 |
| InventoryDataService.find | 语法 | public InventoryBillPO find(String ID) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| InventoryDataService.update | 语法 | public ResultMessage update (InventoryBillPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| InventoryDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### PurchaseDataService

1. 模块概述

PurchaseDataService提供进货（进货退货）数据集体载入、保存、增加、删除、查找

1. 模块内部类的接口规范

PurchaseDataService类的接口规范如下表所示

**PurchaseDataService类接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PurchaseDataService.insert | 语法 | public ResultMessage insert(PurchasePO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| PurchaseDataService.find | 语法 | public PurchasePO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| PurchaseDataService.delete | 语法 | public ResultMessage delete (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| PurchaseDataService.update | 语法 | public ResultMessage update (PurchasePO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| PurchaseDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### RecordDataService

1. 模块概述

RecordDataService提供销售明细表、经营历程表、经营情况表的查看

1. 模块内类接口规范

RecordDataService类的接口规范如下表所示

**RecordDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| RecordDataService.getSaleDetail | 语法 | public ArrayList<PersistentObject> getSaleDetail  (String item) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 返回符合上述条件的商品销售记录，包含如下信息：时间（精确到天），商品名，型号，数量，单价，总额 |
| RecordDataService.getBusinessProcess | 语法 | public ArrayList<PersistentObject> getBusinessProcess  (String item) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 返回1. 销售类单据（销售出货单，销售退货单） 2. 进货类单据（进货单，进货退货单） 3. 财务类单据（付款单，收款单，现金费用单）4. 库存类单据（报溢单，报损单，赠送单）。 |
| RecordDataService.getBusinessCondition | 语法 | public ArrayList<PersistentObject> getBusinessCondition  (String item) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 返回信息： 1. 收入类：销售收入、商品类收入（商品报溢收入 成本调价收入 进货退货差价 代金券与实际收款差额收入）。收入类显示折让后总收入，并显示折让了多少。2. 支出类：销售成本、商品类支出（商品报损 商品赠出）。支出类显示总支出。3. 利润：折让后总收入-总支出 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### AccountBillDataService

1. 模块概述

AccountBillDataService提供收款单、付款单、现金单的集体载入、保存、查看修改操作

1. 模块内类的接口规范

AccountBillDataService类的接口规范如下表所示

**AccountBillDataService类的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| FundDataService.insert | 语法 | public ResultMessage insert  (AccountBillPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| FundDataService.find | 语法 | public AccountBillPO find(String ID) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| FundDataService.delete | 语法 | public ResultMessage delete  (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| FundDataService.update | 语法 | public ResultMessage update  (AccountBillPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| FundDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### ApprovalDataService

1. 模块概述

ApproalDataService提供需要审批的单据的集体载入、保存、查看修改操作

1. 模块内部类的接口

**ApproalDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ApprovalDataService.update | 语法 | public ResultMessage update(PersistentObject po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| ApprovalDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### UserDataService

1. 模块概述

提供用户集体载入、保存、增删改查服务

1. 模块内部类的接口

**UserDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserDataService.insert | 语法 | public ResultMessage insert(UserPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| UserDataService.find | 语法 | public UserPO find(String username) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| UserDataService.delete | 语法 | public ResultMessage delete (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| UserDataService.update | 语法 | public ResultMessage update (UserPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| UserDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### CommodityDataService

1. 模块概述

提供商品集体载入、保存、增删改查服务

1. 模块内部类的接口

**CommodityDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommodityDataService.insert | 语法 | public ResultMessage insert(CommodityPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| CommodityDataService.find | 语法 | public ResultMessage find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| CommodityDataService.delete | 语法 | public ResultMessage delete (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| CommodityDataService.update | 语法 | public ResultMessage update (CommodityPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| CommodityDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### CommoditySortDataService

1. 模块概述

提供商品分类数据集体载入、保存、增删改查服务

1. 模块内部类的接口

**CommoditySortDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySortDataService.insert | 语法 | public ResultMessage insert  (CommoditySortPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| CommoditySortDataService.find | 语法 | public CommoditySortPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| CommoditySortDataService.delete | 语法 | public ResultMessage delete  (String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| CommoditySortDataService.update | 语法 | public ResultMessage update  (CommoditySortPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| CommoditySortDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| CommoditySortDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### SaleDataService

1. 模块概述

提供销售（销售退货）数据集体载入、保存、增加、删除、查找

1. 模块内部类的接口

**SaleDataService的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| SaleDataService.insert | 语法 | public ResultMessage insert(SalesPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| SaleDataService.find | 语法 | public SalesPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| SaleDataService.delete | 语法 | public ResultMessage delete (String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| SaleDataService.update | 语法 | public ResultMessage update (SalesPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| SaleDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### AccountDataService

1. 模块概述

提供账户数据集体载入、保存、增删改查服务

1. 模块内部类的接口

**AccountDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountDataService.insert | 语法 | public ResultMessage insert(AccountPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| AccountDataService.find | 语法 | public AccountPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| AccountDataService.delete | 语法 | public ResultMessage delete (String name) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| AccountDataService.update | 语法 | public ResultMessage update (AccountPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| AccountDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### ClientDataService

1. 模块概述

提供客户数据集体载入、保存、增删改查服务

1. 模块内部类的接口

**ClientDataService类接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ClientDataService.insert | 语法 | public ResultMessage insert(ClientPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| ClientDataService.find | 语法 | public ClientPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| ClientDataService.delete | 语法 | public ResultMessage delete (String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| ClientDataService.update | 语法 | public ResultMessage update (ClientPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| ClientDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### AccountaInitDataService

1. 模块概述

提供期初建账数据集体载入、保存、增加、查看服务

1. 模块内部类的接口

**AccountaInitDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountaInitDataService.insert | 语法 | public ResultMessage insert(AccountaInitPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| AccountaInitDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| AccountaInitDataService.getAllMsg | 语法 | public ArrayList<AccountaInitPO> getAllMsg (); |
| 前置条件 | 无 |
| 后置条件 | 返回全部AccountaInitPO持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### PromotionDataService

1. 模块概述
2. 模块内部类的接口

# 依赖视角

下图是客户端和服务器端各自的包之间的依赖关系



图 1 客户端包图

图 2 服务器端包图